On the Weickian Model in the Context of Open Source Software Development: Some Preliminary Insights

Federico Iannacci
Department of Information Systems, London School of Economics,
Houghton Street, London WC2A 2AE United Kingdom (UK),
F.Iannacci@lse.ac.uk
WWW home page: http://personal.lse.ac.uk/iannacci/

Abstract. Despite being regarded as a path-breaking model of organising, Weick's Enactment-Selection-Retention (ESR) model has been labelled too abstract a model find any practical applications. This paper attempts to show that exploration-oriented open source projects represent valuable case studies where Weick's ESR model can be applied. By taking the Linux case study as a case in point, it is argued that a qualitative analysis of micro interactions (i.e. double interacts) might reveal broad organising patterns. Preliminary implications in terms of coordination and knowledge making processes are discussed in the final section.

1 Introduction

Despite being regarded as a path-breaking model of organising (Tsoukas 1998), Weick's Enactment-Selection-Retention (ESR) model has been labelled too abstract a model to find any practical applications (Aldrich 1999, Harrison 1994). Commenting on Weick's ESR model, Aldrich (1999: 56), maintains, for instance, that "some theorists argue that organizational actors essentially create the context to which they react, thus creating a closed explanatory loop. Not every theorist goes that far, but the concept of enactment –that actions precede interpretation and interpretations create a context for action- places heavy demands on anyone conducting research on why people and organizations behave as they do". By the same token, Harrison (1994: 252- 253) remarks that "valuable as this perspective may be, it is important to recognize that interdependence is never manifested or experienced in quite such abstract terms. Weick's model is an important conceptual tool for understanding how coherent patterns of organization emerge from ongoing sequences of interlocked behaviours, but it retains an unreal, skeletal quality because most of the cultural, situational, and historical contexts associated with these processes have been stripped away... Thus Weick's model is essentially a framework without specific content".

The purpose of this paper is to show that a minimalist approach to organising as advocated by Weick's ESR model can go a long way in terms of explaining the interaction patterns emerging within exploration-oriented open source projects in general and the Linux kernel development in particular. My argument unfolds in the

Please use the following format when citing this chapter:
Iannacci, F., 2006, in IFIP International Federation for Information Processing, Volume 203, Open Source Systems, eds. Damiani, E., Fitzgerald, B., Scacchi, W., Scotto, M., Succi, G., (Boston: Springer), pp. 3-8

following fashion: section two introduces Weick's ESR model in the context of open source software development, section three elaborates on the research methodology, section four identifies a few patterns characterising the Linux kernel organising process and, finally, section five highlights some preliminary insights stemming from my analysis of the Linux case study.

2 On Weick's ESR model in the context of open source software development

Weick's ESR model offers a compelling rationale for understanding why interdependence is a processual accomplishment within any social settings (Harrison 1994). Due to the lack of self-sufficiency, individuals engage in interlocked communicative behaviours to meet their goals. Each individual needs the instrumental communicative act of another individual to perform his consummatory act. However, the enactment of these interlocked behaviours produces equivocality because people utter words that can plausibly be interpreted in two or more ways: the crux of organising then consists of reducing equivocality (i.e. misunderstanding) by means of sensible communication cycles so as to achieve a situation where shared understanding is attained[1].

According to the ESR model, organising is a "consensually validated grammar for reducing equivocality" (Weick 1979: 3) because people literally need to share the same representation (i.e. cause map) of the words they have uttered to act collectively regardless of their individual goals (Weick 1979).

At this level of abstraction, Weick's model seems to have no bearing on technology unless one takes technology to stand for "intensive technology" (Thompson 1967) which feeds back on the social endeavour of organisational actors by constantly disrupting their activities through equivocal displays (Cf. Weick 1979: 22). Hence, for sake of clarification, I take enactment to stand both for social construction of technology and bracketing. By social construction of technology I mean a process whereby developers engage in social interactions to generate software artefacts which were not out there initially; by bracketing, otherwise, I intend to refer to a process whereby the equivocal displays stemming from the emergent technological construct are punctuated and made sense of. In addition, I take selection to stand for numerous decision premises (March and Simon 1958) that serve as assembly rules, that is shared criteria whereby only a subset of social interactions is chosen out of the pool of all communication cycles. Finally, I take retention to stand

[1] Equivocality stands for a situation where given an output message (e.g. a word being uttered or written), there are multiple perceived inputs (i.e. meanings that might have generated that output message) or vice versa (i.e. given an input there are many associated outputs). See Weick (1979: 179-187) on this point. For sake of simplification, in this paper I take equivocality and ambiguity to be synonyms. I also take knowledge and information to be synonyms.

for the stock of knowledge or information retained from the past that can be brought to bear on present decisions. The ESR model so described is outlined below:

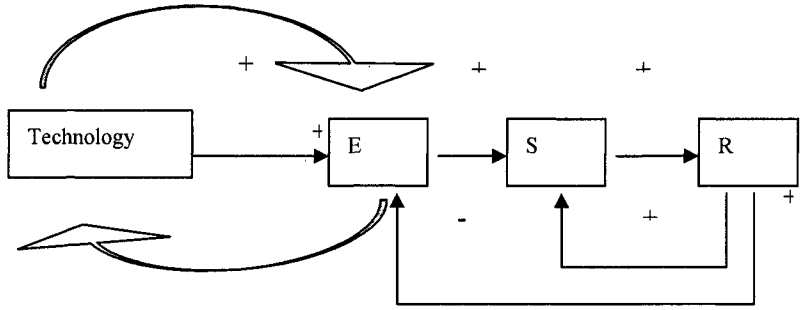

Fig. 1. The ESR Model in Open Source Software Development. Adapted from Weick (1979)

Note that while selection credits the past on the basis of stored memory rules, that is rules stored from past knowledge, enactment discredits the past because it relies on playful behaviours that relax the rules by treating memory as an enemy (March 1988), thus fostering exploration of the space of possibilities. This, in turn, implies that the social construction of technology and, therefore, social interactions are only partially shaped by the stock of past knowledge (i.e. the system's memory) to the extent that developers resort to shared procedures to select subsets of interaction cycles. According to Weick (1979), a system that is simultaneously crediting and discrediting its past is a self-stabilising system because it is able to balance the antithetical pressures deriving from flexibility (i.e. exploration) and stability (i.e. exploitation).

3 On the research methodology

The case study is a research strategy which focuses on understanding the dynamics present within single settings (Eisenhardt 1989). This paper examines the Linux case study to pinpoint broad interaction patterns that apply to the category of exploration-oriented open source projects of which Linux is representative (Nakakoji *et al.* 2002)[2].

Linux is a Unix-like operating system started by Linus Torvalds in 1991 as a private research project. Between 1991 and 1994 the project size burgeoned to the point that in 1994 Linux was officially released as version 1.0. It is now available free

[2] Nakakoji et al. (2002) contend that exploration-oriented projects, including the Linux kernel, aim at pushing the frontline of software development collectively through the sharing of innovations. Contributions made by the community at large exist as feedback and are incorporated only if they are consistent with the ideas of the project leader.

to anyone who wants it and is constantly being revised and improved in parallel by an increasing number of volunteers.

Like many other open source projects, Linux exhibits feature freezes from time to time whereby its leader announces that only bug fixes (i.e. corrective changes) will be accepted in order to enhance the debugging process and obtain a stable release version. The Linux kernel development process, therefore, may be decomposed into a sequence of feature freeze cycles each signalling the impending release of a stable version.

Given my concern with Weick's idea of organising, I set out to use a longitudinal case study (Pettigrew 1990) as my research design. Several feature freezes were analysed spanning the period 1995-2005. Since February 2002 represents a point of rupture in the lifespan of the Linux kernel development process due to the official adoption of BitKeeper (BK), a proprietary version control tool, by Torvalds, I analysed with particular focus the events surrounding the October 2002 feature freeze, the first freeze exhibiting the parallel adoption of two versioning tools, namely BK and CVS (i.e. the Concurrent Versions System)[3]. In analysing such events, I decomposed each thread into sets of two contingent responses between two or more developers, thus taking Weick's (1979) double interact as my unit of analysis.

4 An overview of the Linux kernel organising process

I have claimed above that the ESR model may be viewed as a way of conceptualising the organising process where the collective brackets the equivocal displays stemming from the emergent software construct (i.e. enactment), filters such raw data (i.e. selection) and, finally stores them in various storage devices as knowledge or information (i.e. retention). Put differently, the Linux collective (Shaikh and Cornford 2005) may be conceptualised as an organisational mind of sorts where loosely-coupled developers engage into a set of interactions by following specific decision premises that are collectively shared as assembly rules (i.e. procedures, instructions or guides used to organise the process). Thus, every instance of organising consists of sets of interaction cycles or double interacts and assembly rules whereby such cycles are assembled together and sequenced to create knowledge (Iannacci 2003). But what are the assembly rules that the Linux kernel developers follow?

The longitudinal analysis of the interaction cycles occurring on the Linux Kernel Mailing List (i.e. LKML) and other mailing lists suggests that two rules are followed by developers, namely[4]:

a) Rule of enhancement: select those interaction cycles that enhance the quality of the data inputs. The Linux kernel developers enact their programming skills by

[3] Note that, in early April 2005, Torvalds has replaced BK with Git, a GPL-tool that like BK does not rely on a single, centralised repository and maintains a similar workflow for incorporating new patches. See: http://www.linux.org/news/2005/04/21/0012.html

[4] This list of rules is by no means exhaustive since other rules might well apply (Cf. Weick 1979: 114).

following standardised patch submission procedures, as well as standardised bug reporting formats[5]. Standardisation enhances the quality of the data flows and makes them more amenable to sense-making processes occurring across space and time. Consider, for instance, the following patch submitted during the October 2002 feature freeze:

> *>On 5 Oct 2002, Maksim (Max) Krasnyanskiy wrote:*
> *> > Patch #2:*
> *>Why is it so hard to just read the "submitting patches" thing.*
> I did. Long time ago though :)
> *>Don't bother to email me if you can't be bothered to read how to*
> *>submit patches. People do it all the time, and I'm not interested in*
> *>fetching compressed patches from web-sites etc.*
> Sorry about that. I knew you were gonna pull this stuff from BK any way[6].

Since compressed patches are more equivocal than patches submitted the regular way, Torvalds is explicitly asking developers to follow the pre-defined procedures.

b) Rule of personnel: select those interaction cycles that are manned by the most experienced and, therefore, most trustworthy people. To solve the issue of scalability (i.e. "Linus does not scale"), a loosely-coupled social structure has emerged over time whereby Torvalds interacts with a select number of developers, the so called "Trusted Lieutenants", who, in turn, interact with a few trustworthy people, thus creating a complex attentive system tied together by trust (Weick and Roberts 1993). Without trust developers should expend time and effort to verify the reliability of the patches received. Trust operates as an equivocality-reducing mechanism that ensures reliable performance.

5 Concluding remarks

Despite the obvious limitation concerning the generalisability of findings stemming from a single case study, Weick's ESR model can contribute some original ideas to the study of the open source software development process. Not only does Weick's ESR model shed some light on the delicate issue of coordination by showing that coordination (i.e. organising) accounts for stability in a turbulent context where developers can follow their localised interests in a loosely-coupled fashion; it also helps conceptualise the knowledge-making process considering that the raw data stemming from the emergent source code are transformed into information or knowledge on the basis of collectively-shared assembly rules. Further research should

[5] Note that the argument developed above refers to manual rather than automated procedures.

[6] Source: http://www.ussg.iu.edu/hypermail/linux/kernel/0210.0/2396.html. Note that this is a double interact because we have two contingent responses, namely Torvalds' response to Krasnyanskiy's initial message marked with a single arrow (>) and Krasnyanskiy response to Torvalds, the original message being marked with a double arrow (>>).

be devoted to the study of equivocality considering that electronic contexts compound the sense-making problem due to the lack of social context cues.

References

Aldrich, H. E. (1999) *Organizations Evolving,* Sage Publications, London, Thousand Oaks.

Eisenhardt, K. M. (1989) "Building Theories from Case Study Research", *Academy of Management Review,* **14 (4),** pp. 532-550.

Harrison, T. M. (1994) "Communication and Interdependence in Democratic Organizations", *Communication Yearbook 17,* pp. 247-274.

Iannacci, F. (2003) "The Linux Managing Model", *First Monday 8/12; Address: http://www.firstmonday.org/issues/issue8_12/iannacci/index.html, Accessed on 03/12/03.*

March, J. G. (1988) "The Technology of Foolishness" in *Decisions and Organizations,* (March, J. G. ed.) Basic Blackwell Ltd, Oxford.

March, J. G. and H. A. Simon (1958) *Organizations,* John Wiley & Sons, Inc., New York.

Nakakoji, K., Y. Yamamoto, K. Kishida and Y. Ye (2002) "Evolution Patterns of Open-Source Software Systems and Communities". in *International Workshop Principles of Software Evolution,* pp. 76-85,

Pettigrew, A. M. (1990) "Longitudinal Field Research on Change: Theory and Practice", *Organization Science,* **1 (3),** pp. 267-292.

Shaikh, M. and T. Cornford (2005) "Learning/Organizing in Linux:: A Study of the 'Spaces in Between". in *The 27th International Conference on Software Engineering (ICSE 2005). Open Source Application Spaces: 5th Workshop on Open Source Software Engineering,* St. Louis, Missouri, USA, pp. 57-61,

Thompson, J. D. (1967) *Organizations in Action,* Mc Graw-Hill Book Company, New York.

Tsoukas, H. (1998) "Introduction: Chaos, Complexity and Organization Theory", *Organization,* **5 (3),** pp. 291-313.

Weick, K. E. (1979) *The Social Psychology of Organizing,* Addison-Wesley Publishing Company, Menlo Park, California.

Weick, K. E. and K. Roberts (1993) "Collective Mind in Organizations: Heedful Interrelating on Flight Decks", *Administrative Science Quarterly,* **38 (3),** pp. 357-381.

Acknowledgements:

I wish to thank Maha Shaikh and the two anonymous reviewers for their helpful comments. Obviously, I am to blame for any conceptual mistakes.